DIVERSIFY - Ecology-inspired software evolution for diversity emergence

Benoit Baudry, Martin Monperrus
INRIA, France
benoit.baudry@inria.fr
martin.monperrus@univ-lille1.fr

Cendrine Mony
Univ. de Rennes 1, France
cendrine.mony@univ-rennes1.fr

Franck Chauvel, Franck Fleurey
SINTEF, Norway
franck.chauvel@sintef.no
franck.fleurey@sintef.no

Siobhán Clarke
Trinity College Dublin, Ireland
Siobhan.Clarke@scss.tcd.ie

***Abstract*—DIVERSIFY is an EU funded project, which aims at favoring spontaneous diversification in software systems in order to increase their adaptive capacities. This objective is founded on three observations: software has to constantly evolve to face unpredictable changes in its requirements, execution environment or to respond to failure (bugs, attacks, etc.); the emergence and maintenance of high levels of diversity are essential to provide adaptive capacities to many forms of complex systems, ranging from ecological and biological systems to social and economical systems; diversity levels tend to be very low in software systems.**

DIVERSIFY explores how the biological evolutionary mechanisms, which sustain high levels of biodiversity in ecosystems (speciation, phenotypic plasticity and natural selection) can be translated in software evolution principles. In this work, we consider evolution as a driver for diversity as a means to increase resilience in software systems. In particular, we are inspired by bipartite ecological relationships to investigate the automatic diversification of the server side of a client-server architecture. This type of software diversity aims at mitigating the risks of software monoculture. The consortium gathers researchers from the software-intensive, distributed systems and the ecology areas in order to transfer ecological concepts and processes as software design principles.

I. Introduction

Diversity is acknowledged as a crucial element for resilience, sustainability and increased wealth in many domains such as sociology, economy and ecology. In particular, the multiple forms of biodiversity underpin the productivity and stability of ecosystems in the face of perturbations. The relation between biodiversity and resilience of ecosystems has been called the insurance hypothesis [31]: if all species have different responses to changes in the ecosystem, then, the system as a whole is more likely to endure multiple evolutions and perturbations.

With respect to the large body of theoretical and experimental science that emphasizes the need to conserve high levels of diversity in complex systems, the limited amount of diversity in software intensive systems is a major issue [29], [24], [10]. This is particularly critical as these systems integrate multiple concerns, are connected to the physical world through multiple sensors, run eternally and are open to other services and to users. Such systems, also called collaborative adaptive systems (CAS), mirror the complexity of other complex systems. One major challenge to building collaborative adaptive systems is that current software engineering techniques require architects to foresee all possible adaptation situations the system will have to face. However, the inherent open and dynamic nature of collaborative adaptive systems makes such a-priori knowledge impossible.

DIVERSIFY explores diversity as essential software design principle. The fundamental intuition is that a pool of software variants will represent a reservoir in which the system can find adaptation solutions. The resulting improved adaptive capacities should enable the system to face situations that were unforeseen at design time, and improve the global resilience of systems under study.

The introduction of diversity for the construction of dependable software systems started in the 1970's when Chen and Avizienis proposed N-version programming [5], [1] and Randell proposed recovery blocks [25]. Both approaches consider a set of diverse versions of the same software function. Software diversity is also the foundation of many techniques for increasing security. Many randomization techniques aim at diversifying the execution environment on different machines. For example, Lin et al. [16] randomize the data structure layout of a program to generate diverse binaries that are semantically equivalent. Several other pieces of work randomize instruction sets to provide unique execution environments and limit the ability of attackers to inject malicious code [13], [3].

As long as the software variants are created manually, the number of variants is small and the assumption of failure independence tends to be unsatisfied [14]. Consequently, our objective is to propose techniques that automate the creation of diversity in collaborative adaptive systems through ecological foundations [4].

> DIVERSIFY aims at developing mechanisms that favor the emergence of multiple forms of software diversity in collaborative adaptive systems, through automatic transformation and evolution. The expected outcome is a set of software evolution and maintenance principles that spontaneously sustain diversity in collaborative adaptive systems.

Section II summarizes factual data about the project and section III presents the main objectives and the ecological

foundations for software evolution. Section IV discusses the relevance to the CSMR community. Section V summarizes related work and projects.

II. Essential Facts of DIVERSIFY

- **Name and acronym of the project**: DIVERSIFY - Ecology-inspired software diversity for distributed adaptation
- **Source and amount of funding**: EU funding (1,8 M€)
- **List of participants (with names and affiliations)**:
  - **INRIA, France**: S. Allier, O. Barais, B. Baudry, M. Biazzini, J. Bourcier, M. Monperrus, K. Yeboah-Antwi
  - **SINTEF, Norway**: F. Chauvel, F. Fleurey, H. Song
  - **Trinity College Dublin, Ireland**: S. Clarke, V. Nallur
  - **Université de Rennes 1, France**: B. Gauzens, C. Mony
  - **Ecological board**: M. Hutchings (University of Sussex, UK), B. Kunin (University of Leeds, UK), C. Melian (ETH/EAWAG, Switzerland), E. Thébault (CNRS, France)
- **Web site**: www.diversify-project.eu
- **Duration of the project**: 3 years (Feb. 2013 / Jan. 2016)

III. Scientific Objectives and Foundations

DIVERSIFY aims at formalizing and experimenting with new models and for creating and analyzing software diversity in collaborative adaptive systems, based on the ecological concept of biodiversity. The goal is to increase adaptive capacities in the face of unforeseen structural and environmental variations.

DIVERSIFY will converge towards this main objective through the development of the following scientific and technological objectives

Objective 1: provide automatic synthesis mechanisms for the emergence of software diversity in collaborative adaptive systems. These forms of diversity will result from the translation of biodiversity models into software concepts.

Objective 2: provide novel distributed, diversity-driven, adaptation mechanisms in collaborative adaptive systems. These spontaneous exploration mechanisms will result from the transfer of ecological specialization and adaptation dynamics to the software domain.

Objective 3: develop software modeling and monitoring techniques to provide accurate and updated models at runtime and support distributed adaptation in collaborative, distributed and heterogeneous adaptive systems.

Objective 4: simulate and provide experimental evidence of the effects of software diversity on the adaptive capacities of a distributed collaborative adaptive system in the domain of large-scale smart cities.

A. Ecology as Foundation

Biodiversity. Ecologists acknowledge that a loss of diversity increases the vulnerability of a system in the face of changes in the environment. However, the diversity-stability debate [18] is still challenging because of the number of variables that influence this phenomenon. Two elements inspire DIVERSIFY's research about software diversity:

- Biodiversity types: genetic, specific, functional[1], species diversity, ecosystem diversity.
- Measuring biodiversity: many metrics exist that depend on the type of diversity and the granularity of observation.
- Measuring the effects of biodiversity: many approaches quantify the effect of diversity on the system's robustness, productivity, or stability.

Ecological networks. Food webs, also known as trophic networks, are the most significant model to represent the constituents in an ecosystem. Originally proposed by Lindeman [17], this holistic model captures the different species present in the ecosystem, as well as trophic (resource providing) flows that relate these species. These webs are organized according to trophic levels, which indicate the level of a given species in a food chain. Each trophic level also corresponds to a family of functionally consistent species. Intra and inter trophic level relationships model prey-predator relations. Ecosystems also host a large number of pairwise species relationships represented as bipartite networks [7]. Examples of bipartite ecological relationships include plant-pollinator or host-parasite relationships. Section III-B explains how DIVERSIFY plans to investigate how the structure and dynamics of trophic networks can inspire adaptable software architectures that leverage diversity.

Phenotypic plasticity. Plasticity is the ability of individuals of a given species to modify their characteristics in response to changes in their environment [6]. Particular growth forms in plants result for each individual in a network of ramets (shoots) connected by modified horizontal stems through which information can be shared. Plasticity is found in this network because ramets that uptake resources are able to specialize in heterogeneous environmental conditions to improve their efficiency in resource harvesting. Local specialization supports increased fitness at the clonal network level, while foraging ability allows spatial dispersion in more favorable patches. DIVERSIFY investigates plasticity mechanisms to design software adaptation processes in response to heterogeneous and variable environments.

B. Evolution Rules for Diversified Software

In the first period of the project, the consortium is focusing on bi-partite relationships in ecosystems to tackle monoculture in client-server software architectures. This monoculture, as mentioned by M. Stamp [29], characterizes the fact that most server backends run on the same technology. For example, many web servers run on the same operating system, which represents a major potential threat: if attackers find an exploit

[1] *i.e.*, space competitor, nitrogen fixer, etc.

in the system, they can crash all servers running on this system. This is called the BOBE (blow-one blow-everything) effect in cyber-security.

We model the monoculture of client-server architectures with a bi-partite graph illustrated in figure 1. In this graph, the top layer represents client machines, which are connected to server machines (represented in the bottom layer of the graph). The important point here is that all server machines are the same. Such a network is globally functional: all clients are provided services by server machines, *i.e.*, there is a link between each client and a server. However, the network as a whole is also weak: an attack that can destroy one of the servers can destroy all of them, denying service to all clients.

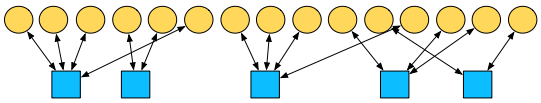

Fig. 1: Software monoculture in client-server relationships

Ecological systems include a large number of bi-partite relationships, such as prey-predator or plant-pollinator [7]. These relationships are also modeled as bi-partite graphs in which nodes represent species and edges represent a specific kind of relationship. For example, figure 2 represents bees and flowers species, as well as the pollinating relationships. Ecological bi-partite networks are very robust to perturbations because of the large diversity of species and the functional redundancy of species in the network. In many cases, if a flower species disappears, most bee species that relied on it for pollen can find pollen in one or two other species. The diversity of flower species increases the chances that the extinction of a particular species species will not lead to the extinction of the others, while the functional redundancy increases the chances that bees can find similar pollens in other flowers.

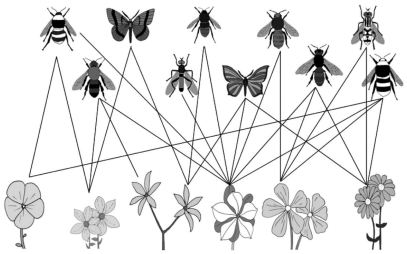

Fig. 2: Example of bi-partite mutualistic relationship

DIVERSIFY leverages ecological results about species diversity in bi-partite relationships to increase diversity in client-server systems. One essential result from ecology is that species diversity and redundancy has emerged through species evolution [20], [11]. DIVERSIFY thus aims at defining software evolution rules, which can lead to the emergence of software diversity in client-server networks. This is illustrated in figure 3. There are different kinds of servers that have emerged through software evolution (*e.g.*, through addition/removal of kernel packages) and client machines are possibly connected to more than one server that can provide the required service. The figure also illustrates that we target a network with more diversity but also with more machines. More servers will provide robustness through redundancy, but it will also increase the cost of the network. The set of software evolution rules, should thus consider a trade-off between the robustness of the network and its global cost.

We adapt state-of-the-art ecological metrics to assess the benefit of diversity of software bi-partite relationships. First, we adapt biodiversity metrics to quantify the evolution of diversity among servers. Shannon entropy is a a classical metric to quantify the diversity of species inside one level of the graph, while several indices exist to measure the diversity of relations [7]. To demonstrate a beneficial effect of diversity on robustness, we adapt the robustness metric of Dunne et al. [8]. This metric for bipartite ecological relations evaluates the ability of one level to survive extinction sequences of the other level. Our experiments will evaluate robustness according to different server extinction sequences, *e.g.*, BOBE attack or random crash.

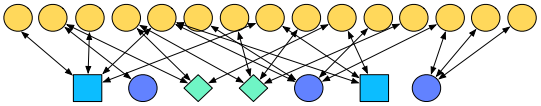

Fig. 3: Emerging software diversity in servers

IV. Relevance to the CSMR community

Software evolution is a major area of research in the CSMR [2] community. This is the essential relationship between the DIVERSIFY project and CSMR. The project aims at identifying software evolution mechanisms that can drive the emergence of software diversity. Examples of software diversity that we aim at providing through evolution include: performance diversity (some maintenance tasks aim at fixing performance issues, but this might also insert new bugs, thus it is worth keeping several versions on different machines), functional diversity (upgrades in the system might include new features, but it might be safe to keep several instances of the old version running while assessing the quality and stability of the new feature [12]) and technology diversity (*e.g.*, when a database fails on one server, it is possible to change the DB management system on this machine only, but not on all others, leveraging market diversity [2]).

V. Related Work

DIVERSIFY is related to two main areas of software engineering: software diversity and component-based adaptation.

Software diversity has previously been investigated for fault tolerance [5], [25] and cyber security [13], [16]. The most recent work focuses on automatic synthesis of software diversity to maximize its potential impact on resilience. R. Feldt [9]

[2]Conference on Software Maintenance and Reverse engineering

used genetic programming to automatically synthesize variants of an aircraft controller in order to achieve failure diversity. M. Rinard and colleagues [28], [26] have developed unsound program transformations that support the runtime production of diversity and handle changes in quality of service. Forrest and colleagues have explored genetic programming for automatic bug fixing [15] and neutral mutation [27]. Recently, Mendez et al. [21] have observed natural diversity in object-oriented API usage.

Software architecture aims at reducing complexity through abstraction and separation of concerns by providing a common understanding of component, connector and configuration [19], [30]. Several works have shown the benefits of this design style for dealing with adaptive systems [23]. However, these models partially failed to gain wide adoption. There are two likely reasons: (1) the models were not accompanied by actual system level facilities for dynamic evolution (too many constraints exist on component implementation for example) and (2) the type of dynamism they supported was in some ways overly constrained. Recently, several component-based platforms try to overcome these limitations to provide abstractions and system level support for rich adaptation primitives.

There are related EU funded projects. CONNECT (Emergent CONNECTor for Eternal software intensive Networked Systems) is a FET-IP that aims at dropping interoperability barriers by synthesizing on the fly the connectors via which networked systems communicate. DIVERSIFY explores the positive side of dynamic interoperability abilities by using them to create diversified systems.

DIVA (Dynamic Variability in complex, Adaptive systems) was a FP7-STREP that explored how to build self-adaptive systems on top of an adaptation model using models@runtime. DiVA resulted in models for large-scale, distributed software adaptation [22]. DIVERSIFY will build on the results of DiVA to exploit emergent diversity.

The SMScom project (ERC Grant) aimed at developing a consistent and integrated set of methods and tools for the design, validation, and operation of self-managing situational software. The term situational indicates that software is built to address a particular situation, problem, or challenge, and behaves according to the evolving situation in which it operates. The emergence of software diversity, tackled by DIVERSIFY, is a specific kind of situational evolution.

REFERENCES

[1] A. Avizienis. The n-version approach to fault-tolerant software. *IEEE Trans. on Software Engineering (TSE)*, (12):1491–1501, 1985.

[2] D. Barman, J. Chandrashekar, N. Taft, M. Faloutsos, L. Huang, and F. Giroire. Impact of it monoculture on behavioral end host intrusion detection. In *Proc. of the workshop on Research on enterprise networking*, pages 27–36. ACM, 2009.

[3] E. G. Barrantes, D. H. Ackley, S. Forrest, and D. Stefanović. Randomized instruction set emulation. *ACM Trans. on Information and System Security (TISSEC)*, 8(1):3–40, 2005.

[4] B. Baudry and M. Monperrus. Towards ecology-inspired software engineering. *CoRR*, 2012.

[5] L. Chen and A. Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. In *Proc. of the Int. Symp. on Fault-Tolerant Computing (FTCS'78)*, pages 3–9, 1978.

[6] H. de Kroons and M. J. Hutchings. Morphological plasticity in clonal plants: the foraging concept reconsidered. *Journal of Ecology*, 83(1):143–152, 1995.

[7] C. F. Dormann, J. Fründ, N. Blüthgen, and B. Gruber. Indices, graphs and null models: Analyzing bipartite ecological networks. *Open Ecology Journal*, 2:7–24, 2009.

[8] J. A. Dunne, R. J. Williams, and N. D. Martinez. Network structure and biodiversity loss in food webs: robustness increases with connectance. *Ecology Letters*, 5(4):558–567, 2002.

[9] R. Feldt. Generating diverse software versions with genetic programming: an experimental study. *IEE Proceedings-Software*, 145(6):228–236, 1998.

[10] S. Forrest, A. Somayaji, and D. H. Ackley. Building diverse computer systems. In *The Sixth Workshop on Hot Topics in Operating Systems (HOTOS'97)*, pages 67–72. IEEE, 1997.

[11] S. Gavrilets and J. B. Losos. Adaptive radiation: contrasting theory with data. *Science*, 323(5915):732–737, 2009.

[12] P. Hosek and C. Cadar. Safe software updates via multi-version execution. In *Proc. of the Int. Conf. on Software Engineering (ICSE)*, pages 612–621, 2013.

[13] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proc. of the conf. on Computer and communications security (CCS)*, pages 272–280, 2003.

[14] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Trans. on Software Engineering (TSE)*, (1):96–109, 1986.

[15] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer. Genprog: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38:54–72, 2012.

[16] Z. Lin, R. D. Riley, and D. Xu. Polymorphing software by randomizing data structure layout. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 107–126. Springer, 2009.

[17] R. L. Lindeman. The trophic-dynamic aspect of ecology. *Ecology*, 23(4):399–417, 1942.

[18] K. S. McCann. The diversity–stability debate. *Nature*, 405(6783):228–233, 2000.

[19] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *Software Engineering, IEEE Transactions on*, 26(1):70–93, 2000.

[20] C. J. Melián, D. Alonso, D. P. Vázquez, J. Regetz, and S. Allesina. Frequency-dependent selection predicts patterns of radiations and biodiversity. *PLoS computational biology*, 6(8):e1000892, 2010.

[21] D. Mendez, B. Baudry, and M. Monperrus. Empirical Evidence of Large-Scale Diversity in API Usage of Object-Oriented Software. In *Proc. of the Int. Conf. on Source Code Analysis and Manipulation (SCAM)*, pages –, Netherlands, 2013.

[22] B. Morin, O. Barais, G. Nain, and J.-M. Jézéquel. Taming dynamically adaptive systems using models and aspects. In *Proc. of the Int. Conf. on Software Engineering (ICSE)*, pages 122–132, 2009.

[23] P. Oreizy, N. Medvidovic, and R. N. Taylor. Runtime software adaptation: framework, approaches, and styles. In *Companion of the 30th international conference on Software engineering*, pages 899–910. ACM, 2008.

[24] D. L. Parnas. Which is riskier: Os diversity or os monopoly? *Communications of the ACM*, 50(8):112, 2007.

[25] B. Randell. System structure for software fault tolerance. *IEEE Trans. on Software Engineering (TSE)*, (2):220–232, 1975.

[26] M. C. Rinard. Obtaining and reasoning about good enough software. In *Proc. of the Design Automation Conference (DAC)*, pages 930–935, 2012.

[27] E. Schulte, Z. Fry, E. Fast, W. Weimer, and S. Forrest. Software mutational robustness. *Genetic Programming and Evolvable Machines*, pages 1–32, 2013.

[28] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proc. of the Symp. on Foundations of software engineering (FSE'11)*, ESEC/FSE '11, pages 124–134, 2011.

[29] M. Stamp. Risks of monoculture. *Communications of the ACM*, 47(3):120, 2004.

[30] R. Van Ommering, F. Van Der Linden, J. Kramer, and J. Magee. The koala component model for consumer electronics software. *Computer*, 33(3):78–85, 2000.

[31] S. Yachi and M. Loreau. Biodiversity and ecosystem productivity in a fluctuating environment: the insurance hypothesis. *Proceedings of the National Academy of Sciences*, 96(4):1463–1468, 1999.